Lecture 3: Linear Classification
Last time: Image Classification

assume given set of discrete labels
{dog, cat, truck, plane, ...}

→ cat
k-Nearest Neighbor

training set

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
test images
Linear Classification

1. Define a **score function**

   class scores
Linear Classification

1. define a **score function**

\[ f(x_i, W, b) = W x_i + b \]

data (image)

“weights”

“bias vector”

“parameters”

class scores
Linear Classification

1. **define a score function**
   (assume CIFAR-10 example so 32 x 32 x 3 images, 10 classes)

\[ f(x_i, W, b) = Wx_i + b \]

- **data (image)** [3072 x 1]
- **weights**
- **bias vector**
- **class scores**
1. **define a score function**
(assume CIFAR-10 example so 32 x 32 x 3 images, 10 classes)

$$f(x_i, W, b) = W x_i + b$$

- **data (image)**: \([3072 \times 1]\)
- **weights**: \([10 \times 3072]\)
- **bias vector**: \([10 \times 1]\)
- **class scores**: \([10 \times 1]\)
Linear Classification

\[ f(x_i; W, b) = W x_i + b \]
Interpreting a Linear Classifier

Question:
what can a linear classifier do?

$$f(x_i, W, b) = Wx_i + b$$

[Diagram with matrices and images of animals]
Interpreting a Linear Classifier

Example training classifiers on CIFAR-10:

\[ f(x_i, W, b) = Wx_i + b \]
Interpreting a Linear Classifier

\[ f(x_i, W, b) = Wx_i + b \]
**Bias trick**

$$f(x_i, W, b) = W x_i + b$$

$$f(x_i, W) = W x_i$$
So far:

We defined a (linear) **score function**:

\[
f(x_i, W, b) = Wx_i + b
\]

---

**input image**

- Input pixel values:
  - 0.2, -0.5, 0.1, 2.0
  - 1.5, 1.3, 2.1, 0.0
  - 0, 0.25, 0.2, -0.3

- Weights (`W`):
  - 56
  - 231
  - 24
  - 2

- Bias (`b`):
  - 1.1
  - 3.2
  - -1.2

- Calculated scores:
  - Cat score: -96.8
  - Dog score: 437.9
  - Ship score: 61.95

---

Fei-Fei Li & Andrej Karpathy

Lecture 2 - 13

7 Jan 2015
2. Define a **loss function** (or **cost function**, or **objective**)

2. Define a **loss function** (or cost function, or objective)

- scores, label $y_i$ → loss.

\[ f(x_i, W) \rightarrow L_i \]
2. Define a **loss function** (or cost function, or objective)

- scores, label  $\rightarrow$ loss.

$$f(x_i, W) \quad y_i \quad L_i$$

Example:

$$f(x_i, W) = [13, -7, 11]$$

$$y_i = 0$$
2. Define a **loss function** (or cost function, or objective)

- scores, label \( f(x_i, W) \) \( y_i \) \( L_i \) \( \rightarrow \) **loss.**

**Example:**

\[
f(x_i, W) = [13, -7, 11] \\
y_i = 0
\]

**Question:** if you were to assign a single number to how “unhappy” you are with these scores, what would you do?
2. Define a **loss function** (or cost function, or objective)  
One (of many ways) to do it: **Multiclass SVM Loss**

\[
L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)
\]
2. Define a **loss function** (or cost function, or objective)
One (of many ways) to do it: **Multiclass SVM Loss**

\[
L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)
\]

(One possible generalization of Binary Support Vector Machine to multiple classes)

\[
L_i = C \max(0, 1 - y_i w^T x_i) + R(W)
\]
2. Define a **loss function** (or cost function, or objective)
One (of many ways) to do it: **Multiclass SVM Loss**

\[
L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)
\]

- loss due to example \(i\)
- sum over all incorrect labels
- difference between the correct class score and incorrect class score
\[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \]

- loss due to example \( i \)
- sum over all incorrect labels
- difference between the correct class score and incorrect class score
Example: \[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \]

\[ f(x_i, W) = [13, -7, 11] \]
\[ y_i = 0 \]

loss = ?
Example:  \[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \]

\[ f(x_i, W) = [13, -7, 11] \]
\[ y_i = 0 \]

\[ L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10) \]
\[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \]

def L_i_vectorized(x, y, W):
    scores = W.dot(x)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i
There is a bug with the objective…

\[
L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right]
\]
L2 Regularization

\[ R(W) = \sum_k \sum_l W_{k,l}^2 \]

\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]

Regularization strength
L2 regularization: motivation

\[ x = [1, 1, 1, 1] \]

\[ w_1 = [1, 0, 0, 0] \]

\[ w_2 = [0.25, 0.25, 0.25, 0.25] \]

\[ w_1^T x = w_2^T x = 1 \]
\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]
\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]

Do we have to cross-validate both \( \Delta \) and \( \lambda \)?
\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]

\[ \max(0, \delta f + \Delta) \]
\[ \rightarrow \max(0, \delta f + \Delta_2) \quad \text{if } \Delta \rightarrow \Delta_2 \]
\[ \rightarrow \max(0, \frac{\Delta_2}{\Delta} \delta f + \Delta_2) \quad \text{then scale the weights by } \frac{\Delta_2}{\Delta} \]
\[ = \max(0, \frac{\Delta_2}{\Delta} (\delta f + \Delta)) \]
\[ = \frac{\Delta_2}{\Delta} \max(0, \delta f + \Delta) \]
So far…

1. Score function

\[ f(x_i, W, b) = WX_i + b \]

2. Loss function

\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]
The diagram shows a neural network architecture. It includes nodes labeled with variables and functions, with arrows indicating the flow of information and loss calculation.

Key elements:
- \( W \): Weight matrix
- \( x_i \): Input data
- \( y_i \): Output data
- \( f(x_i, W) \): Score function
- \( L \): Loss function
- Regularization loss
- Data loss

The network processes an input \( x_i \) through a score function \( f(x_i, W) \) and calculates the loss to adjust the weights \( W \).
Softmax Classifier

$$f(x_i, W) = W x_i$$

score function

is the same

(extension of Logistic Regression to multiple classes)
Softmax Classifier

\[ f(x_i, W) = W x_i \]

score function is the same

\[ L_i = - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]
Softmax Classifier

\[ f(x_i, W) = W x_i \]

score function is the same

\[ L_i = - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]

softmax function

i.e. we’re minimizing the negative log likelihood.
Softmax Classifier

\[ f(x_i, W) = W x_i \]

score function is the same

\[ L_i = - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]

softmax function
Softmax Classifier

\[ f(x_i, W) = W x_i \]

score function is the same

\[ L_i = - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]

i.e. we’re minimizing the negative log likelihood.

\[ [1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26] \]
matrix multiply + bias offset

\[
\begin{pmatrix}
0.01 & -0.05 & 0.1 & 0.05 \\
0.7 & 0.2 & 0.05 & 0.16 \\
0.0 & -0.45 & -0.2 & 0.03 \\
\end{pmatrix}
\begin{pmatrix}
-15 \\
22 \\
-44 \\
56 \\
\end{pmatrix}
+ \begin{pmatrix}
0.0 \\
0.2 \\
-0.3 \\
\end{pmatrix}
\]

hinge loss (SVM)

\[
\max(0, -2.85 - 0.28 + 1) + \\
\max(0, 0.86 - 0.28 + 1)
= 1.58
\]

cross-entropy loss (Softmax)

\[
\begin{pmatrix}
-2.85 \\
0.86 \\
0.28 \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0.058 \\
2.36 \\
1.32 \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0.016 \\
0.631 \\
0.353 \\
\end{pmatrix}
= 0.452
\]
Softmax vs. SVM

- Interpreting the probabilities from the Softmax

\[ [1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26] \]
Softmax vs. SVM

- Interpreting the probabilities from the Softmax

\[
[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]
\]

suppose the weights \( W \) were only half as large

(we use a higher regularization strength)
Softmax vs. SVM

- Interpreting the probabilities from the Softmax

\[ [1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26] \]

Suppose the weights \( W \) were only half as large:

\[ [0.5, -1, 0] \rightarrow [e^{0.5}, e^{-1}, e^0] = [1.65, 0.37, 1] \rightarrow [0.55, 0.12, 0.33] \]
Softmax vs. SVM

- Interpreting the probabilities from the Softmax

\[ [1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26] \]

Suppose the weights \( W \) were only half as large:

\[ [0.5, -1, 0] \rightarrow [e^{0.5}, e^{-1}, e^0] = [1.65, 0.37, 1] \rightarrow [0.55, 0.12, 0.33] \]

What happens in the limit, as the regularization strength goes to infinity?
\[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1) \]

\[ L_i = -\log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]

scores:

\[
\begin{align*}
[10, -2, 3] \\
[10, 9, 9] \\
[10, -100, -100]
\end{align*}
\]
\[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1) \]

\[ L_i = -\log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]
Interactive Web Demo time....

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/
Summary

- We introduced a **parametric approach** to image classification
- We defined a **score function** (linear map)
- We defined a **loss function** (SVM / Softmax)

One problem remains: How to find $W,b$?
Next class: Optimization, Backpropagation

Find the $W,b$ that minimizes the loss function.